Capitolo 13

Il package dell'AWT

di David R. Chung, rivisto da Christopher Burdess

IN QUESTO CAPITOLO

Il package dell'Abstract Windowing Toolkit (AWT) è una libreria di classi di Java che possono essere utilizzate per creare interfacce utente grafiche (GUI, Graphical User Interface), le quali permettono agli utenti di interagire con l'applicazione o con l'applet di Java nello stesso modo in cui sono abituati a interagire con altre applicazioni nelle loro piattaforme native.

Poiché Java è un linguaggio per tutte le piattaforme, l'AWT non implementa tutte le funzionalità di tutti i sistemi operativi specifici con GUI, ma solamente le funzionalità comuni a tutti i sistemi. La programmazione con l'AWT pertanto è una soluzione per le interfacce utente che garantisce al massimo la portabilità.

Le classi dell'AWT possono essere suddivise nelle seguenti tre categorie:

- ✔ componenti;
- ✔ layout;
- ✔ classi di utilità.

In questo capitolo vengono esaminate tutte queste categorie e forniti esempi di come incorporare le classi dell'AWT nelle applicazioni e negli applet di Java. Inoltre, il capitolo presenta una discussione dei *peer* e dei loro vantaggi e limiti.

Componenti

Conosciuti anche come elementi di controllo dell'interfaccia utente, o *controlli*, i *componenti* sono diversi elementi dell'interfaccia che possono essere visualizzati all'utente e che possono anche essere programmati in modo da rispondere all'interazione dell'utente tramite il mouse o la tastiera. Questa *gestione degli eventi* è una parte importante della programmazione della GUI e viene discussa più avanti nel paragrafo: "Eventi".

La maggior parte dei componenti dell'AWT deriva dalla classe `Component`; l'eccezione più importante è costituita dalla classe `MenuComponent` e dalle relative sottoclassi, che vengono discusse più avanti.

La classe `Component` è una classe astratta che definisce elementi comuni a tutti i componenti: il carattere, il colore, la visualizzazione, la modifica delle forme e la gestione degli eventi.

Le classi astratte *sono classi che dichiarano uno o più metodi come* `abstract`. *I metodi astratti devono essere ridefiniti da una sottoclasse; le classi astratte non possono essere istanziate.*

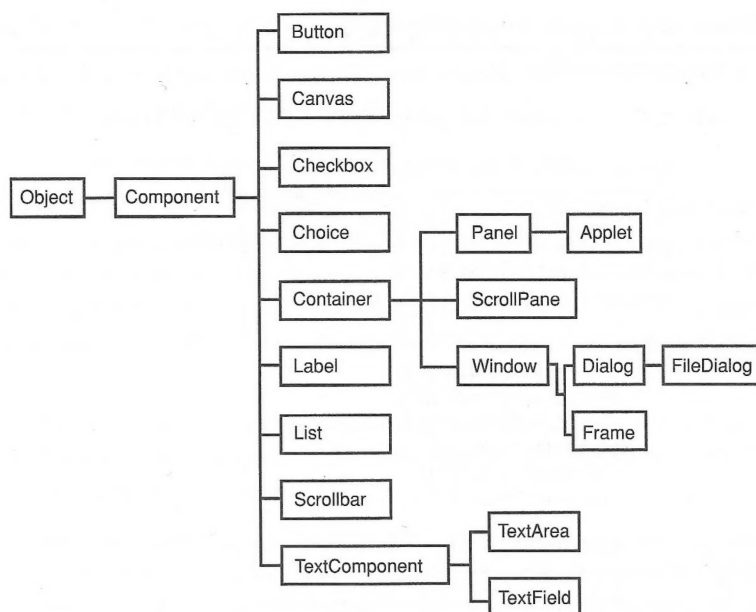
Nella Figura 13.1 è mostrata la struttura della classe `Component`. Tutte le classi appartengono al package `java.awt`, a eccezione della classe `Applet`, che fa parte del package `java.applet`.

Una classe di un package può derivare da una superclasse di un altro package. Pertanto, la classe `Applet` *del package* `java.applet` *deriva dalla classe* `Panel` *del package* `java.awt`. *Un package non indica la gerarchia di classe, ma è solamente una convenzione per riunire certi tipi di funzionalità. L'aspetto più importante dal punto di vista funzionale degli oggetti* `Applet` *è che possono essere incorporati in documenti Web e interpretati da un browser Web in base a certe regole di sicurezza, non che fanno parte di un'interfaccia utente grafica di un programma di Java.*

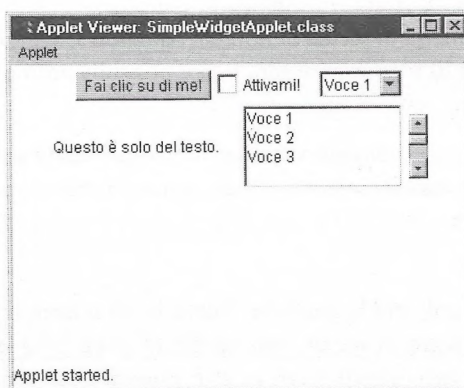
I componenti possono essere suddivisi in ulteriori quattro gruppi funzionali principali:

- componenti semplici;
- componenti di testo;
- contenitori;
- la classe `Canvas`.

In questo capitolo vengono anche discussi i menu. Nonostante non derivino dalla classe `Component` e non condividano alcune proprietà dei componenti, ad esempio la posizione, la larghezza, il colore e così via, i menu hanno altre proprietà simili ai componenti, in particolare i caratteri e la gestione degli eventi. In ogni caso, si farà riferimento agli elementi menu come a componenti, in quanto sono elementi visibili dell'interfaccia utente. Per ulteriori dettagli, si veda il paragrafo: "Menu" più avanti in questo capitolo.

Figura 13.1
L'albero della classe Component.

Controlli semplici

I controlli semplici *sono componenti per cui l'interazione con l'utente richiesta è minima o nulla e che non contengono nessun altro tipo di componente. I controlli semplici includono le classi* `Button`, `Checkbox`, `Choice`, `Label`, `List` *e* `Scrollbar`. *Nella maggior parte dei sistemi operativi con GUI esistono correlazioni di questi componenti. L'applet* `SimpleWidgetApplet` *nella Figura 13.2 mostra i controlli semplici dell'AWT nella piattaforma Microsoft Windows 95. Il codice per l'applet* `SimpleWidgetApplet` *è incluso nel Listato 13.1 e si trova anche nel CD-ROM allegato al libro.*

Figura 13.2
L'applet SimpleWidgetApplet.

Listato 13.1 *L'applet SimpleWidgetApplet.*

```
import java.awt.*;

public class SimpleWidgetApplet extends java.applet.Applet {

     Button button = new Button("Fai clic su di me!");
     Checkbox checkbox = new Checkbox("Attivami!");
     Choice choice = new Choice();
     Label label = new Label("Questo è solo del testo.");
     List list = new List();
     Scrollbar scrollbar = new Scrollbar();

     public void init() {
          choice.add("Voce 1");
          choice.add("Voce 2");
          choice.add("Voce 3");
          list.add("Voce 1");
          list.add("Voce 2");
          list.add("Voce 3");

          add(button);
          add(checkbox);
          add(choice);
          add(label);
          add(list);
          add(scrollbar);
     }
}
```

Innanzitutto, vengono creati dei controlli. Alcuni di essi hanno dei costruttori semplici che possono essere utilizzati per impostare i valori iniziali; ad esempio, alla classe `Button` può essere passata una stringa nel costruttore per inizializzare l'etichetta del pulsante. Altri componenti non hanno costruttori utili, di norma perché non si conosce il numero di valori che deve essere passato, come nel caso dei componenti `Choice` e `List`. In questo caso, si utilizza il costruttore predefinito e si impostano le proprietà successivamente. Con il controllo `Choice` nell'applet `SimpleWidgetApplet` è stato utilizzato il metodo `add(`*`String`*`)` per impostare le stringhe da visualizzare nel menu di scelta. Si noti che, quando si aggiungono elementi in un elenco (e più avanti nei menu) in questo modo, non è possibile ordinarli: le stringhe vengono visualizzate nell'ordine in cui le si aggiunge.

Questi componenti sono stati quindi aggiunti all'applet utilizzando il metodo `add(`*`Component`*`)` dell'applet stesso. A volte è possibile saltare un passaggio creando e aggiungendo contemporaneamente un componente:

```
add(new Button("Ciao!"));
```

A volte questo approccio è utile per le etichette. Tuttavia, di norma si preferisce mantenere un riferimento all'oggetto creato, come in `SimpleWidgetApplet`. In questo modo è possibile fare facilmente riferimento agli oggetti creati in altri metodi della classe, in particolare nei metodi di gestione degli eventi, così da poterne esaminare e impostare le proprietà.

Le funzionalità fornite dai controlli semplici sono le seguenti.

- Il componente `Label`, come menzionato, viene utilizzato semplicemente per visualizzare del testo all'utente, che non può fare nulla con esso.
- Il componente `Button` viene utilizzato per iniziare un'azione associata.
- Il componente `Checkbox` viene utilizzato per rappresentare lo stato di una variabile `boolean` (`true` o `false`, selezionata o deselezionata). Le caselle di controllo possono essere associate a un particolare oggetto `CheckboxGroup`, costruendole con esso o utilizzando il metodo `setCheckboxGroup(`*`CheckboxGroup`*`)`. In un gruppo di caselle di controllo è possibile selezionare una sola casella alla volta; se una è selezionata, tutte le altre sono deselezionate.
- Il componente `Choice` viene utilizzato per selezionare una tra diverse opzioni stringa. Viene mostrata solo l'opzione selezionata, mentre le altre vengono visualizzate in un elenco a discesa quando si fa clic sul componente `Choice`.
- Il componente `List` può essere utilizzato nello stesso modo del componente `Choice`, vale a dire per selezionare una tra diverse opzioni stringa. In più, un componente `List` può contenere diverse selezioni, cosa che è possibile specificare nel costruttore oppure utilizzando il metodo `setMultipleMode(`*`boolean`*`)`. Nel caso di componenti `List` che permettono la selezione multipla, l'utente può selezionare contemporaneamente diverse opzioni.
- Il componente `Scrollbar` viene utilizzato per rappresentare un valore numerico. Di norma, questo valore viene utilizzato per rappresentare la quantità visibile di un componente in un contenitore. Per ulteriori dettagli, si faccia riferimento alla discussione sulla classe `ScrollPane` nel paragrafo: "Contenitori", più avanti in questo capitolo.

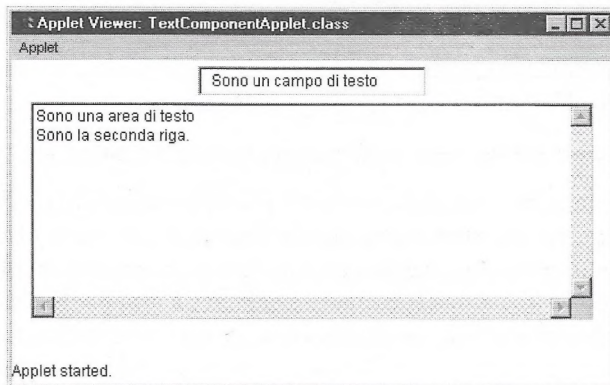
Componenti di testo

Nelle applicazioni o negli applet di Java è possibile aggiungere due componenti dell'AWT in cui l'utente può immettere del testo tramite la tastiera. Questi componenti di testo sono chiamati `TextField` e `TextArea` e sono entrambi sottoclassi di `TextComponent` (Figura 13.1).

`TextComponent` è una classe di base astratta che incapsula molte caratteristiche comuni ai due componenti. La classe `TextField` è un semplice componente per l'inserimento di testo su un'unica riga, mentre la classe `TextArea` supporta più righe e lo scorrimento verticale. Entrambi i componenti possono essere utilizzati per visualizzare valori stringa inseriti dall'utente. Nella Figura 13.3 è mostrato un esempio. Il codice dell'applet `TextComponentApplet` è incluso nel Listato 13.2 e si trova anche nel CD-ROM allegato al libro.

Listato 13.2 *L'applet TextComponentApplet.*

```
import java.awt.*;
public class TextComponentApplet extends java.applet.Applet {
     TextField textfield = new TextField("Sono un campo di testo");
     TextArea textarea = new TextArea("Sono una area di testo\nSono la seconda riga.");
```

Figura 13.3 *L'applet TextComponentApplet.*

```
        public void init() {
            add(textfield);
            add(textarea);
        }
    }
```

Come si può vedere in questo esempio, i componenti di testo hanno dei pratici costruttori che possono essere utilizzati per specificare le stringhe da visualizzare inizialmente. È anche possibile impostare altre proprietà dei componenti di testo: sia `TextField` *che* `TextArea` *permettono di specificare il numero di colonne (caratteri) e inoltre* `TextArea` *permette di impostare il numero di righe.*

Contenitori

Un *contenitore* è un componente che può contenerne altri. Tutti i contenitori nell'AWT derivano dalla classe astratta `Container`. I contenitori principali sono `Panel`, `ScrollPane` e `Window`.

I *pannelli* vengono di norma utilizzati per raggruppare componenti in aree specifiche dello schermo. Poiché a ogni componente può essere attribuito un layout diverso, i pannelli sono utili se si desidera posizionare i componenti con un layout diverso da quello utilizzato dal contenitore genitore (di norma `Window` o `Applet`). Un `Applet` è un tipo di pannello con ulteriori proprietà relative alle implementazioni del browser; la classe `Applet` del package `java.applet` è l'unica classe derivata dall'AWT che non fa parte del package `java.awt` o di uno dei suoi sottopackage.

I *pannelli scorrevoli* sono un tipo di pannello. È possibile aggiungere un componente figlio al contenitore `ScrollPane` che può avere dimensioni superiori a quelle del contenitore `ScrollPane` stesso. In questo caso, il pannello scorrevole è dotato di barre di scorrimento orizzontali e/o verticali che permettono all'utente di visualizzare aree diverse del componente figlio.

La classe `Window` incapsula una finestra di alto livello e rappresenta una finestra senza barra del titolo o bordo. Le sottoclassi di questa classe forniscono i titoli, le immagini delle icone e le funzioni di controllo della finestra da parte dell'utente, ad esempio i comandi per la chiusura e le funzionalità di ingrandimento e di riduzione a icona. Queste sottoclassi sono `Frame` e `Dialog`.

È possibile utilizzare la classe `Window` *come superclasse di base per finestre a comparsa e componenti, ad esempio per le bandierine.*

La classe `Frame` fornisce tutte le funzionalità per le finestre di un'applicazione nel sistema operativo nativo, a eccezione del fatto che non è possibile accedere all'handle del sistema operativo o al PID della finestra (o di altre finestre dell'ambiente). I frame sono la base per la maggior parte delle applicazioni GUI di Java; poiché implementano l'interfaccia `MenuContainer`, possono presentare una barra dei menu (si veda il paragrafo: "Menu" più avanti in questo capitolo).

La classe `Dialog` rappresenta una finestra che di norma viene visualizzata per un breve periodo per mostrare o per richiedere informazioni transitorie. Se sono necessarie delle informazioni per un passaggio in un'operazione (ad esempio "Il file è stato modificato. Si desidera salvarlo?"), che non saranno più necessarie dopo il passaggio, si dovrebbe utilizzare una finestra di dialogo. Nello stesso modo, se un'applicazione deve informare l'utente che è stato eseguito un passaggio ed è necessaria una risposta che indichi che l'utente ha capito il messaggio, si dovrebbe utilizzare una finestra di dialogo. Un oggetto `Dialog` può essere dichiarato come *modale* durante la creazione o richiamando `setModal(true)`. Una *finestra di dialogo modale* non permette l'interazione dell'utente con la finestra genitore finché non è chiusa.

La classe `FileDialog` rappresenta un tipo speciale di finestra di dialogo che richiede all'utente un percorso o un nome di file. Poiché i percorsi e i nomi di file vengono specificati in modo diverso in base al sistema operativo, questa classe viene utilizzata per evitare i problemi causati dal dover soddisfare tutte le possibili specifiche.

Si noti che tutte le finestre, quando vengono create, sono inizialmente invisibili. Per renderle visibili, è necessario richiamare il metodo `show()`. Di norma, questo si fa dopo aver aggiunto tutti i componenti, a meno che non si stiano sviluppando componenti dinamici.

Per un esempio di come funzionano i frame e le finestre di dialogo, si osservi l'applicazione `DialogFrame` descritta nel paragrafo: "Eventi" più avanti in questo capitolo.

Menu

I componenti menu, come già menzionato, non derivano dalla classe `Component`, ma dalla classe astratta `MenuComponent`. Nella Figura 13.4 è mostrata la gerarchia di `MenuComponent`.

La classe `MenuBar` *incapsula il concetto di menu aggiunto a un* `Frame`*. È possibile impostare la* `MenuBar` *per un* `Frame` *richiamando il metodo* `setMenuBar(MenuBar)`*, come indicato nell'applicazione* `MenuFrame` *rappresentata nella Figura 13.5 e nel Listato 13.3, incluso anche nel CD-ROM allegato al libro.*

Figura 13.4 *L'albero della classe MenuComponent.*

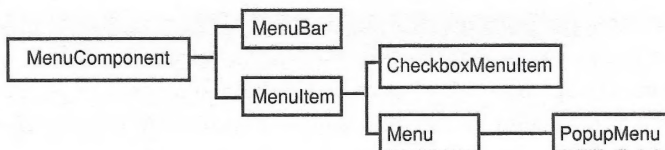
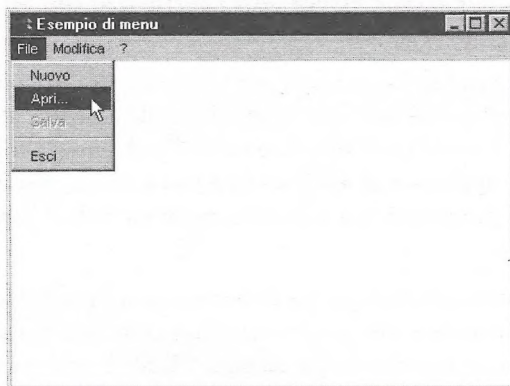



Figura 13.5 *L'applicazione MenuFrame.*

Listato 13.3 *L'applicazione MenuFrame.*

```
import java.awt.*;

public class MenuFrame extends Frame {
     MenuItem fileNew = new MenuItem("Nuovo");
     MenuItem fileOpen = new MenuItem("Apri...");
     MenuItem fileSave = new MenuItem("Salva");
     MenuItem fileExit = new MenuItem("Esci");
     MenuItem editUndo = new MenuItem("Annulla");
     MenuItem editCut = new MenuItem("Taglia");
     MenuItem editCopy = new MenuItem("Copia");
     MenuItem editPaste = new MenuItem("Incolla");
     MenuItem helpContents = new MenuItem("Sommario");
     MenuItem helpAbout = new MenuItem("Informazioni su MenuFrame...");

     public MenuFrame() {
          super("Esempio di menu");

          MenuBar menubar = new MenuBar();
          Menu fileMenu = new Menu("File");
          Menu editMenu = new Menu("Modifica");
          Menu helpMenu = new Menu("?");

          fileMenu.add(fileNew);
          fileMenu.add(fileOpen);
          fileSave.setEnabled(false);
          fileMenu.add(fileSave);
          fileMenu.addSeparator();
          fileMenu.add(fileExit);
```

```
            editUndo.setEnabled(false);
            editMenu.add(editUndo);
            editMenu.addSeparator();
            editCut.setEnabled(false);
            editMenu.add(editCut);
            editCopy.setEnabled(false);
            editMenu.add(editCopy);
            editPaste.setEnabled(false);
            editMenu.add(editPaste);

            helpMenu.add(helpContents);
            helpMenu.addSeparator();
            helpMenu.add(helpAbout);

            menubar.add(fileMenu);
            menubar.add(editMenu);
            menubar.add(helpMenu);
            menubar.setHelpMenu(helpMenu);
            setMenuBar(menubar);
            setSize(new Dimension(400, 300));
            show();
        }

        public static void main(String[] args) {
            MenuFrame me = new MenuFrame();
        }
    }
```

Oltre a specificare l'etichetta da visualizzare per ogni elemento di menu, la classe `MenuItem` permette anche di creare il menu con una chiave che può attivare il `MenuItem` specifico.

Come si può vedere, il codice del Listato 13.3 disattiva gli elementi del menu non applicabili. In questo breve esempio, ciò è ridondante in quanto non vi sarà mai del contenuto nell'oggetto `MenuFrame` da salvare o da modificare. Gli elementi del menu sono stati disattivati solo a scopo dimostrativo. Inoltre, in realtà non accade nulla quando vengono selezionati gli elementi del menu, perché all'applicazione non è stata ancora aggiunta alcuna funzionalità di gestione degli eventi. Per un esempio di applicazione che risponde all'interazione dell'utente, si veda il paragrafo: "Eventi" più avanti in questo capitolo.

Menu a comparsa

I *menu a comparsa*, utilizzati per fornire funzioni contestuali per l'oggetto sottostante a un componente, sono diventati molto comuni in piattaforme quali Windows 95, NT 4.0 e CDE. La classe di base per creare i menu a comparsa è `PopupMenu`, derivata da `MenuComponent`. Con questa classe è possibile creare un menu a comparsa e visualizzarlo in relazione al componente di riferimento per mezzo del metodo `show(`*`Component, int, int`*`)`. Le due variabili `int` nel metodo `show()` rappresentano le coordinate x e y (in pixel) relative al componente specificato. Il menu a comparsa deve essere aggiunto a un componente valido, che può essere quello da cui si specifica la posizione oppure un altro.

Uno dei problemi correlati ai menu a comparsa è l'evento utilizzato per attivare il menu, che varia da piattaforma a piattaforma. Ad esempio, in Motif il menu si apre a seguito di un evento di pressione del pulsante del mouse, mentre in Windows il menu viene visualizzato a seguito di un evento di rilascio del pulsante. La classe `MouseEvent` ha una funzione speciale `isPopupTrigger()` per gestire questo problema.

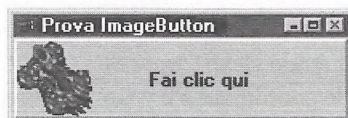
La classe Canvas e i contesti Graphics

`Canvas` è una classe generica di componenti che può essere utilizzata per implementare metodi grafici. Di per sé, la classe `Canvas` è simile alla classe `Panel`, ma non è un contenitore e non è possibile aggiungervi altri componenti. In sostanza, si tratta semplicemente di una classe `Component` da cui è possibile creare un'istanza. I canvas vengono derivati per visualizzare immagini, per disegnare grafici e per creare componenti personalizzati quali barre di progresso, cartelle con schede e pulsanti personalizzati. È possibile ottenere tutto ciò ridefinendo il metodo `paint(`*`Graphics`*`)`. Con il contesto `Graphics` fornito, è possibile richiamare metodi grafici per disegnare in un canvas punti, linee, testo e immagini.

L'oggetto `Graphics` permette l'incapsulamento delle informazioni sullo stato necessarie per i diversi metodi di rappresentazione utilizzati da Java, ad esempio il colore corrente, il tipo di carattere e l'origine degli assi. Le coordinate passate ai metodi `Graphics` sono considerate relative a questa origine; i metodi di rappresentazione richiamati sull'oggetto `Graphics` modificano solo i pixel all'interno dei bordi del riquadro.

Tra i metodi più utili forniti da questo oggetto vi sono `drawImage()`, utilizzato per rappresentare un'immagine nel contesto grafico, `drawLine()`, utilizzato per disegnare linee, `drawOval()`, utilizzato per disegnare ellissi e cerchi, `drawString()`, utilizzato per disegnare testo, `fillRect()`, utilizzato per disegnare rettangoli pieni, `getFontMetrics()`, che richiama un oggetto `FontMetrics` con informazioni sul carattere corrente e che può essere utilizzato per calcolare dove disegnare il testo, e `setFont()` e `setColor()`, utilizzati per selezionare i colori e i caratteri per disegnare gli oggetti. Per ulteriori informazioni sui caratteri, sui colori e sulle immagini, si veda il paragrafo: "Classi di utilità" più avanti in questo capitolo.

La classe `ImageButton` mostra come creare sottoclassi di `Canvas` per disegnare un pulsante che può visualizzare un'immagine e del testo. La Figura 13.6 mostra un oggetto `ImageButton` aggiunto a un oggetto `Frame` nella piattaforma Windows 95. Il Listato 13.4 include il codice utilizzato per fare tutto questo (il codice si trova anche nel CD-ROM allegato al libro).

Figura 13.6
Il componente ImageButton in un frame.

Listato 13.4 *La classe ImageButton.*

```
import java.awt.*;
public class ImageButton extends Canvas {

     private String label;
     private Image image;

     public ImageButton() {
          this(null, null);
     }

     public ImageButton(String label) {
          this(label, null);
     }

     public ImageButton(Image image) {
          this(null, image);
     }

     public ImageButton(String label, Image image) {
          this.label = label;
          this.image = image;
     }

     public String getLabel() {
          return label;
     }

     public Image getImage() {
          return image;
     }

     public void setLabel(String label) {
          if(!label.equals(this.label)) {
               this.label = label;
               repaint();
          }
     }

     public void setImage(Image image) {
          if(!image.equals(this.image)) {
               this.image = image;
               repaint();
          }
     }

     public void paint(Graphics g) {
          Color b = getBackground();
          Font f = g.getFont();
          Rectangle r = g.getClipBounds();

          super.paint(g);

          // Disegna l'immagine.
          // La scala in modo da farla rientrare nel rettangolo di ritaglio
          // e centrarla in verticale.
```

```
            if(image != null && r.width>4 && r.height>4) {
                 Dimension d = new Dimension(image.getWidth(this),
                              image.getHeight(this));
                 float scale = 1;
                 float fdwidth = (new Integer(d.width)).floatValue();
                 float frwidth = (new Integer(r.width)).floatValue();
                 float fdheight = (new Integer(d.height)).floatValue();
                 float frheight = (new Integer(r.height)).floatValue();
                 if((fdwidth*scale) > (frwidth*scale))
                      scale = scale * (frwidth/fdwidth);
                 if((fdheight*scale) > (frheight*scale))
                      scale = scale * (frheight/fdheight);
                 int ndwidth = (new Float(fdwidth * scale)).intValue() - 4;
                 int ndheight = (new Float(fdheight * scale)).intValue() - 4;
                 int yOffset = (r.height - ndheight) / 2;
                 g.drawImage(image, 2, yOffset, ndwidth, ndheight, this);
            }

            // Disegna l'etichetta.
            // La centra nel rettangolo di ritaglio.
            if(label!=null) {
                 FontMetrics fm = g.getFontMetrics();
                 g.drawString(label, (r.width - fm.stringWidth(label)) / 2, (r.height/2)
                                     + (fm.getAscent()/2));
            }

            // Disegna lo smusso del pulsante.
            // Occorre disegnare le linee due volte per aumentare lo spessore.
            g.setColor(b.brighter());
            g.drawLine(0, r.height-1, 0, 0);
            g.drawLine(1, r.height-2, 1, 1);
            g.drawLine(0, 0, r.width-1, 0);
            g.drawLine(1, 1, r.width-2, 1);
            g.setColor(b.darker());
            g.drawLine(r.width-1, 0, r.width-1, r.height-1);
            g.drawLine(r.width-2, 1, r.width-2, r.height-2);
            g.drawLine(r.width-1, r.height-1, 0, r.height-1);
            g.drawLine(r.width-2, r.height-2, 1, r.height-2);
       }
  }
```

Nella routine `paint()`, le variabili `height` e `width` fanno riferimento all'altezza e alla larghezza di `ImageButton`. Questi numeri sono superiori di 1 rispetto all'indice dell'ultimo pixel nel riquadro del pulsante; i pixel sono infatti numerati da 0 a `width-1` e da 0 a `height-1`, come la lunghezza degli array, che è sempre maggiore di 1 rispetto all'indice dell'ultimo elemento.

La classe `ImageButton` non incapsula un vero pulsante su cui è possibile fare clic per attivare degli eventi. Per trasformarla in un pulsante selezionabile, è necessario definire una variabile istanza booleana (ad esempio `down`) che abbia un flag quando il pulsante riceve un evento di pressione del pulsante del mouse e che non lo abbia quando riceve un evento di rilascio del pulsante. La routine `paint()` può quindi esaminare la variabile `down` e disegnare il pulsante in modo appropriato. Ad esempio, è possibile scambiare le chiamate a `g.setColor()` nella routine dello smusso per invertire quest'ultimo, oppure spostare l'immagine e il testo a

destra e in basso di una determinata distanza, per dare all'utente un'indicazione del fatto che il pulsante è stato selezionato. È possibile implementare la funzionalità di Windows, che fa in modo che gli sfondi dei pulsanti appaiano circondati da un riquadro quando viene fatto clic su di essi.

Per dare vita all'oggetto `ImageButton`, è necessario che sia in grado di gestire gli eventi. La gestione degli eventi viene discussa nel prossimo paragrafo.

Eventi

La *gestione degli eventi* è una delle preoccupazioni maggiori degli sviluppatori di GUI. L'altra è la gestione del layout, che viene discussa nel paragrafo: "Gestione del layout" più avanti in questo capitolo. Per fortuna, con l'AWT la gestione degli eventi è relativamente semplice da apprendere e da implementare. Prima dell'API del JDK 1.1, si utilizzava un modello di eventi ereditario che necessitava di un considerevole supporto e che era poco pratico da utilizzare. Nella versione corrente del JDK; l'AWT utilizza un modello di eventi di delegazione, più robusto e flessibile, che permette un maggiore controllo al momento della compilazione.

Quando l'utente inizia un evento, ad esempio facendo clic sul mouse o digitando qualcosa con la tastiera, l'evento viene generato e reso disponibile per l'applicazione o per l'applet di Java. Nel seguito viene spiegato come intercettare ed elaborare questi eventi.

Ascoltatori e adattatori

Un evento si propaga da un oggetto sorgente (componente) a un oggetto "ascoltatore". L'oggetto sorgente è l'oggetto della GUI su cui l'utente ha fatto clic, in cui ha digitato del testo o con cui ha interagito in qualche modo. Gli ascoltatori sono oggetti che si registrano come interessati in certi tipi di eventi su una sorgente; possono essere altri oggetti della GUI od oggetti separati responsabili degli eventi delegati: in quest'ultimo caso, è possibile scrivere codice di gestione degli eventi che separa gli elementi della GUI dalla loro funzionalità. Nell'AWT, gli eventi che vengono passati di norma derivano dalla classe `AWTEvent`.

L'AWT presenta due tipi di eventi concettualmente diversi: di basso livello e di semantica. Gli *eventi di basso livello* si occupano dell'input specifico dell'utente o degli eventi a livello del sistema di finestre; gli *eventi di semantica*, invece, non si occupano dei dettagli di quale interazione è avvenuta, ma del significato dell'azione. Ad esempio, quando si preme il tasto Invio dopo aver digitato qualcosa in un campo di testo, si vuole eseguire un'azione, come convalidare il contenuto del campo di testo; allo stesso modo, quando si fa clic su un pulsante o doppio clic su un elemento di un elenco, si desidera eseguire un'azione. Tuttavia, quando si fa clic su una barra di scorrimento o quando se ne trascina la casella, si desidera regolare un valore. Questo dimostra come l'evento di basso livello (il clic con il mouse) esegua compiti diversi per componenti diversi, e questo è il motivo per cui sono necessari gli eventi di semantica.

Gli eventi di basso livello e di semantica sono entrambi sottoclassi della classe `AWTEvent`, che è una sottoclasse di `java.util.EventObject`. `AWTEvent` si trova nel package `java.awt`. Tutti gli altri eventi discussi sono parte del package `java.awt.event`. Nella Figura 13.7 è mostrata la gerarchia di classe per gli eventi di basso livello.

In realtà vi è un ulteriore evento di basso livello, `PaintEvent`, che di norma viene utilizzato solo internamente e che non è progettato per l'utilizzo con il modello di ascoltatori. La Figura 13.8 mostra la struttura di classi per gli eventi di semantica.

La classe `ActionEvent` incapsula la nozione di "fare qualcosa" o di "eseguire un'azione"; gli altri due eventi sono più specifici: la classe `AdjustmentEvent` rappresenta una variabile numerica di regolazione, mentre la classe `ItemEvent` indica che lo stato di un elemento è cambiato.

In base a questo modello, tutto ciò che deve fare una classe è implementare una determinata interfaccia di ascolto e registrarsi come interessata a un oggetto sorgente. Tuttavia, gli ascoltatori per gli eventi di basso livello sono progettati per ascoltare diversi tipi di eventi (ad esempio, `WindowListener` ascolta gli eventi di attivazione, di chiusura, di disattivazione, di ingrandimento, di riduzione a icona e di apertura delle finestre). Se una classe implementa questa interfaccia, è necessario fornire i metodi che gestiscono questi eventi.

Per rendere più semplici le cose, il modello di eventi dell'AWT include classi di adattatori *per gli ascoltatori di eventi di basso livello. Queste classi, che si trovano nel package* `java.awt.event`, *forniscono l'implementazione predefinita di tutti i metodi, in modo che sia possibile scegliere quali ridefinire nel codice. Poiché il corpo del metodo adattatore è vuoto, l'utilizzo degli adattatori equivale a utilizzare direttamente le interfacce di ascolto, con la differenza che non è necessario specificare ogni metodo in quest'ultima.*

Figura 13.7
L'albero degli eventi di basso livello.

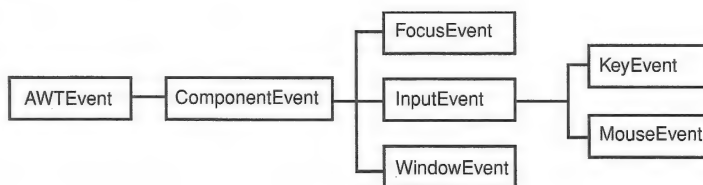
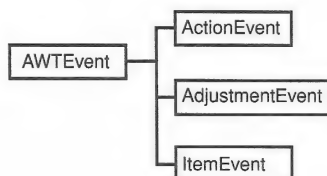



Figura 13.8
L'albero degli eventi semantici.

La Figura 13.9 mostra la gerarchia di adattatori e di ascoltatori di eventi di basso livello. La Figura 13.10 mostra le interfacce di ascolto degli eventi di semantica. Tutti i ricevitori ereditano l'interfaccia `java.util.EventListener` e sono inclusi nel package `java.awt.event`.

La coda di eventi

L'AWT fornisce una coda degli eventi del sistema in attesa, rappresentata dalla classe `java.awt.EventQueue`. Questa classe contiene un metodo statico per restituire la coda di eventi del sistema: `getEventQueue()`. È possibile utilizzare questa coda per conoscere in anteprima gli eventi destinati per gli oggetti nel sistema e, cosa più importante, è possibile inserire nella coda nuovi eventi.

Ciò comporta problemi per la sicurezza: nonostante le applicazioni possano utilizzare liberamente le code di eventi, se si permette agli applet di manipolare liberamente gli eventi, si potrebbero avere serie implicazioni. Pertanto, il metodo `getEventQueue()` è protetto da un `SecurityManager` che non permette agli applet non fidati di accedere direttamente alla coda. Esistono invece dei metodi appropriati di `Applet` per osservare la coda. Questi metodi sono ristretti, in modo che gli applet possano accedere solo agli eventi che avvengono nei componenti all'interno della loro gerarchia.

Figura 13.9 *L'albero di ascoltatori e adattatori di eventi di basso livello.*

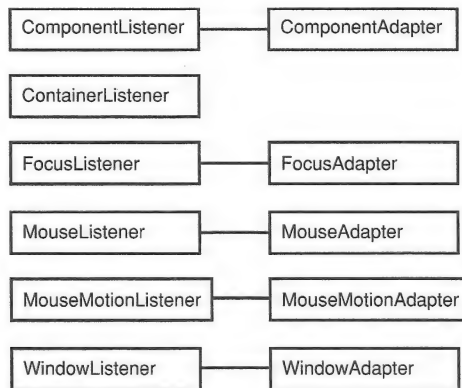
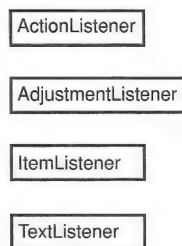



Figura 13.10 *Ascoltatori di eventi semantici.*

Gestione degli eventi

L'applicazione `DialogFrame` del Listato 13.5 (inclusa anche nel CD-ROM allegato al libro) mostra l'utilizzo di diversi metodi e oggetti evento e il funzionamento delle classi `Frame` e `Dialog`.

Listato 13.5 *L'applicazione DialogFrame.*

```
import java.awt.*;
import java.awt.event.*;

public class DialogFrame extends Frame implements ActionListener, WindowListener {

     Dialog dialog;
     Button openDialogButton;
     Button closeDialogButton;
     Button closeFrameButton;

     public DialogFrame() {
          super("Esempio per finestre di dialogo e frame");
          setLayout(new FlowLayout());
          addWindowListener(this);

          openDialogButton = new Button("Apri finestra");
          openDialogButton.addActionListener(this);

          closeFrameButton = new Button("Chiudimi");
          closeFrameButton.addActionListener(this);

          add(openDialogButton);
          add(closeFrameButton);

          pack();
          show();
     }

     public void showDialog() {
          dialog = new Dialog(this, "Ecco la finestra di dialogo", true);
          dialog.setLayout(new FlowLayout());
          dialog.addWindowListener(this);

          closeDialogButton = new Button("Chiudi finestra");
          closeDialogButton.addActionListener(this);

          dialog.add(closeDialogButton);

          dialog.pack();
          dialog.show();
     }

     public void actionPerformed(ActionEvent e) {
          String buttonCommand = e.getActionCommand();
          if(buttonCommand.equals("Apri finestra"))
               showDialog();
          else if(buttonCommand.equals("Chiudi finestra"))
               dialog.dispose();
```

```
        else if(buttonCommand.equals("Chiudimi"))
            processEvent(new WindowEvent(this, WindowEvent.WINDOW_CLOSING));
    }

    public void windowClosing(WindowEvent e) {
        Window originator = e.getWindow();
        if(originator.equals(this)) {
            this.dispose();
            System.exit(0);
        } else if(originator.equals(dialog))
            dialog.dispose();
    }

    public void windowActivated(WindowEvent e) { }
    public void windowDeactivated(WindowEvent e) { }
    public void windowDeiconified(WindowEvent e) { }
    public void windowClosed(WindowEvent e) { }
    public void windowIconified(WindowEvent e) { }
    public void windowOpened(WindowEvent e) { }

    public static void main(String[] args) {
        DialogFrame me = new DialogFrame();
    }
}
```

Innanzitutto ci si assicura che la classe `DialogFrame` sia in grado di ricevere eventi dalle sorgenti a cui è interessata, vale a dire i clic del mouse sui pulsanti (che sono eventi semantici) e gli eventi delle finestre (in modo da poter intercettare gli eventi di basso livello di chiusura delle finestre: diversamente l'utente non potrebbe terminare l'applicazione chiudendo la finestra). Pertanto `DialogFrame` implementa le interfacce `ActionListener` e `WindowListener` e include metodi per gestire gli eventi che riceve.

Quando viene creata, `DialogFrame` per prima cosa richiama il supercostruttore della classe `Frame` con un argomento stringa che imposta la stringa specificata come titolo della finestra. La riga successiva imposta il layout (si veda il paragrafo: "Gestione del layout" più avanti in questo capitolo). Il metodo successivo, `addWindowListener(`*`WindowListener`*`)`, si registra in `DialogFrame` in modo da ascoltare gli eventi delle finestre. Quindi il codice crea il pulsante da inserire nella cornice e si registra come `ActionListener` per gli eventi su questo pulsante. Poiché le finestre sono sempre invisibili quando vengono create, il codice di seguito richiama il metodo `show()` per visualizzare la cornice. `showDialog()` crea una finestra di dialogo modale e registra `DialogFrame` come suo ascoltatore. Il metodo crea anche un pulsante nella finestra di dialogo e registra `DialogFrame` come ascoltatore di azione per il pulsante.

Vengono quindi implementati i metodi di gestione degli eventi per `DialogFrame`. Innanzitutto si implementa il metodo `actionPerformed(`*`ActionEvent`*`)`, che gestisce l'evento clic sul pulsante. Viene valutato su quale pulsante è stato fatto clic e si esegue l'azione correlata. In questo esempio, il metodo `getActionCommand()` di `ActionEvent` viene utilizzato per restituire l'etichetta del pulsante. Come alternativa, si sarebbe potuto utilizzare il metodo `getSource()` di `ActionEvent` per restituire l'oggetto sorgente e confrontarlo con i riferimenti dei pulsanti (`openDialogButton` e così via). Facendo clic su `openDialogButton` si richiama `showDialog()`

per visualizzare la finestra di dialogo; facendo clic su `closeDialogButton` si richiama `dispose()` sulla finestra di dialogo, nascondendola e rimuovendo tutte le risorse del sistema utilizzate per gestirla. Facendo clic su `closeFrameButton`, tuttavia, non si chiude direttamente il frame, ma si invia un nuovo evento di chiusura della finestra. Questo evento viene intercettato da `DialogFrame`, che agisce da ascoltatore di eventi della finestra, e viene elaborato come se fosse una richiesta di chiusura della finestra (cosa che in realtà è). Perché ci si dovrebbe preoccupare di eseguire tutto questo processo tortuoso di invio di eventi, quando la chiusura della finestra e la chiamata a `System.exit()` sono semplici e dirette? In questo caso, tutta l'elaborazione necessaria viene eseguita dalla routine di chiusura dell'ascoltatore della finestra, ma quando si sviluppano applicazioni più complesse, spesso è necessaria ulteriore elaborazione prima che l'applicazione possa terminare. Se si inizia a duplicare codice in metodi diversi e se successivamente si deve modificare in qualche modo il codice, è necessario considerare tutte le altre parti di codice che forniscono la stessa funzionalità. Molti programmatori esperti scrivono metodi speciali, ad esempio `closeAndExit()`, che vengono richiamati da qualsiasi routine per chiudere l'applicazione, assicurandosi in questo modo che vi sia un solo modo in cui il sistema può terminare.

Poiché `DialogFrame` è un'applicazione e non un applet, invece di richiamare `processEvent()`, si sarebbe potuto richiedere un handle alla coda di eventi del sistema e inviarvi l'evento di chiusura del sistema:

```
EventQueue.getEventQueue().postEvent(new WindowEvent(this,
                    WindowEvent.WINDOW_CLOSING));
```

Tuttavia, poiché `processEvent()` fa sì che l'evento venga gestito immediatamente, questo metodo di solito è più utile.

Di seguito è necessario ridefinire tutti i metodi rilevanti in `WindowListener` per gestire gli eventi delle finestre. Si confronta la finestra sorgente con i riferimenti a `DialogFrame` e alla finestra di dialogo figlia, quindi si chiudono le finestre di conseguenza. È inoltre necessario fornire implementazioni vuote degli altri metodi di `WindowListener`, in quanto non si sta utilizzando un oggetto adattatore separato.

Si noti che, poiché questa è un'applicazione e non un applet, è necessario includere un metodo `main()` della seguente forma:

```
public static void main(String[])
```

Il parametro array `String` fa riferimento a tutti i parametri della riga di comando passati all'applicazione. Dopo aver compilato questa classe, è possibile eseguirla con un interprete Java dalla riga di comando nel seguente modo:

```
java DialogFrame
```

Gli eventi Focus e Keyboard

Fuoco è un termine utilizzato per descrivere quale componente deve ricevere gli eventi della tastiera. Poiché nella stessa finestra possono essere visibili diversi componenti, ognuno potenzialmente in grado di gestire gli eventi della tastiera, il fuoco viene utilizzato per deter-

minare la destinazione degli eventi della tastiera. Ad esempio, se si desidera ricevere gli eventi della tastiera su un oggetto `Frame` o `Canvas`, il componente deve prima richiedere il fuoco utilizzando il metodo `requestFocus()`.

L'AWT include lo spostamento del fuoco senza mouse, che viene implementato premendo il tasto [Tab] per spostarsi in avanti o i tasti [Maiusc]+[Tab] per spostarsi indietro. Alcuni componenti (quelli che si basano sui peer della piattaforma sottostante) hanno questa capacità innata. Ad esempio, se si sviluppano nuovi componenti derivati da `Canvas` che sono destinati a ricevere eventi della tastiera, è necessario ridefinire il metodo `isTabbable()` per restituire `true` e catturare l'evento di pressione del pulsante del mouse sul componente, in modo che venga richiamato `requestFocus()`. La maggior parte dei progettisti di interfacce utenti implementa inoltre altre routine di disegno quando il componente ha il fuoco; un pulsante, ad esempio, può avere una riga tratteggiata all'interno dello smusso.

Funzioni degli Appunti

La maggior parte degli utenti delle GUI si aspetta di essere in grado di trasferire dati, quali testo e immagini, da un'applicazione all'altra utilizzando le funzionalità di taglia, copia e incolla degli Appunti. L'API per gli oggetti trasferibili si trova nell'interfaccia `java.awt.datatransfer.Transferable`. Gli oggetti che implementano questa interfaccia devono fornire una serie di formati in cui specificare i dati esistenti. Il package `java.awt.datatransfer` include una classe per il tipo di dati più comuni: la classe `StringSelection` implementa l'interfaccia `Transferable`.

La classe `Clipboard` incapsula gli Appunti in Java. È possibile creare tutti gli oggetti `Clipboard` desiderati per scopi privati; tuttavia, un solo oggetto, `System`, viene utilizzato per l'interfaccia con gli Appunti del sistema e pertanto con applicazioni non Java. È possibile richiamare gli Appunti `System` tramite la chiamata a `getSystemClipboard()` su un oggetto `Toolkit valido`. Per scrivere dati in un oggetto `Clipboard`, è necessario implementare l'interfaccia `ClipboardOwner`. Il trasferimento avviene con due metodi: per scrivere dati negli Appunti, si utilizza il metodo `setContents(`*`Transferable, ClipboardOwner`*`)` di `Clipboard`, mentre per leggere dagli Appunti, si utilizza il metodo `getContents(`*`Object`*`)`.

I parametri *`ClipboardOwner`* e *`Object`* fanno riferimento all'oggetto che ha effettuato la chiamata, mentre gli oggetti *`Transferable`* sono i dati degli Appunti. Quando si legge dagli Appunti, è necessario richiedere un elenco dei formati disponibili dall'oggetto *`Transferable`* e leggere i dati nel formato desiderato con il metodo `getTransferData()` dell'oggetto *`Transferable`*.

Gestione del layout

Finora, in questo capitolo i componenti sono stati aggiunti nei contenitori senza indicare in che modo dovessero essere disposti. L'AWT include un gruppo di classi dette *gestori di layout*, o *layout*, che gestiscono il posizionamento dei componenti. Tutti i layout implementano l'interfaccia `LayoutManager`. Quando si imposta un layout per un contenitore utilizzando il

metodo `setLayout(LayoutManager)` e si aggiunge un componente nel contenitore, si aggiunge anche il componente al layout del contenitore. I layout inclusi nella libreria dell'AWT vengono discussi nel seguito.

Il gestore FlowLayout

Il gestore `FlowLayout` dispone i componenti in righe. Questo layout si rivela maggiormente utile per implementare barre degli strumenti e altri elementi simili. Il gestore `FlowLayout` inserisce tutti i componenti possibili in una riga, prima di iniziare a inserirli nella riga sottostante. La disposizione dei componenti nella riga è determinata dall'ordine in cui essi vengono aggiunti nel contenitore. `FlowLayout` è il layout predefinito per la classe `Panel`, pertanto anche gli applet utilizzano questo gestore di layout, se non ne viene specificato uno diverso.

È possibile specificare la spaziatura tra i componenti di un `FlowLayout` utilizzando i metodi `setHgap()` e `setVgap()`, oppure indicandola al momento della creazione (il valore predefinito è di 5 pixel per entrambi gli spazi orizzontale e verticale). Il gestore `FlowLayout` rispetta gli inset di un componente, che di conseguenza, se specificati, vengono aggiunti allo spazio totale tra i componenti.

È anche possibile specificare l'allineamento del flusso (su quale lato allineare i componenti) nel costruttore o con il metodo `setAlignment()`.

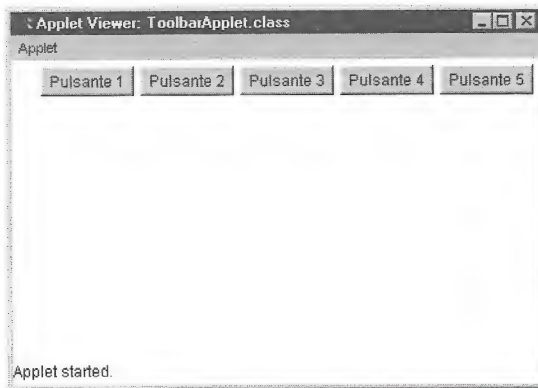
L'applet `ToolbarApplet`*, rappresentato nella Figura 13.11 e incluso nel Listato 13. 6 e nel CD-ROM allegato al libro, visualizza una serie di pulsanti posizionati con un gestore* `FlowLayout` *con allineamento a destra.*

Listato 13.6 *L'applet ToolbarApplet.*

```
import java.awt.*;

public class ToolbarApplet extends java.applet.Applet {

     public void init() {
          setLayout(new FlowLayout(FlowLayout.RIGHT));
          add(new Button("Pulsante 1"));
          add(new Button("Pulsante 2"));
          add(new Button("Pulsante 3"));
          add(new Button("Pulsante 4"));
          add(new Button("Pulsante 5"));
     }
}
```

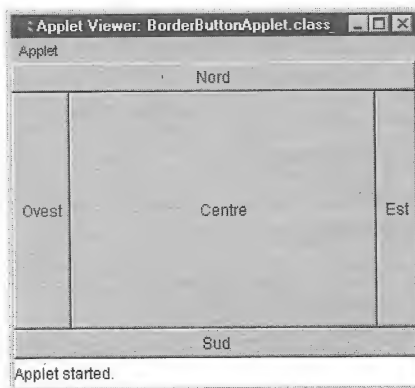
Il gestore BorderLayout

Il gestore `BorderLayout` è un semplice layout che posiziona i controlli in modo che riempiano lo spazio nel contenitore. Quando si aggiunge un componente in un contenitore con il

Figura 13.11
L'applet ToolbarApplet.

layout `BorderLayout`, è anche possibile specificare una delle cinque posizioni sotto forma di stringa: `"North"`, `"East"`, `"South"`, `"West"` e `"Center"`. Le prime quattro specificano su quale lato del contenitore deve essere inserito il componente (nord, est, sud o ovest), mentre la posizione `"Center"` inserisce il componente al centro del contenitore. Se non si specifica la posizione, viene utilizzata quella predefinita `"Center"`. Nel contenitore possono essere visualizzati al massimo cinque componenti, anche se è possibile creare disposizioni più complesse inserendo più contenitori uno dentro l'altro. Il gestore `BorderLayout` è il layout predefinito per la classe `Window`, pertanto i frame e le finestre di dialogo utilizzano questo gestore, se non ne viene specificato uno diverso.

Come con il gestore `FlowLayout`, anche con il gestore `BorderLayout` è possibile specificare gli spazi orizzontali e verticali.

La Figura 13.12 mostra come vengono disposti i componenti con il gestore `BorderLayout`; il Listato 13.7 include il codice utilizzato per creare questo applet. Il codice si trova anche nel CD-ROM allegato al libro.

Figura 13.12
L'applet BorderButtonApplet.

Listato 13.7 *L'applet BorderButtonApplet.*

```
import java.awt.*;

public class BorderButtonApplet extends java.applet.Applet {

     public void init() {
          setLayout(new BorderLayout());
          add(new Button("Nord"), "North");
          add(new Button("Est"), "East");
          add(new Button("Sud"), "South");
          add(new Button("Ovest"), "West");
          add(new Button("Centre"));
     }
}
```

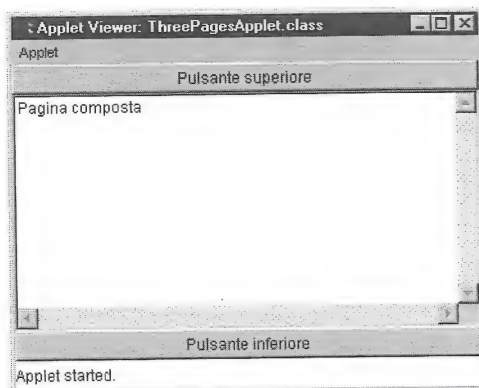
Il gestore CardLayout

Il gestore CardLayout è un layout interessante, nel senso che non cerca di ridimensionare i componenti in modo da visualizzarli tutti nel contenitore, ma li visualizza uno alla volta come in un archivio a schede Rolodex.

Il Listato 13.8 include l'applet ThreePagesApplet*, che si trova anche nel CD-ROM allegato al libro; la Figura 13.13 mostra il contenitore* CardLayout *con il terzo controllo, un pannello con dei pulsanti e del testo.*

Listato 13.8 *L'applet ThreePagesApplet.*

```
import java.awt.*;
import java.awt.event.*;

public class ThreePagesApplet extends java.applet.Applet implements MouseListener {

     Button page1Button;
     Label page2Label;
     TextArea page3Text;
     Button page3Top;
     Button page3Bottom;
     CardLayout layout;

     public void init() {
          setLayout(layout = new CardLayout());

          add(page1Button = new Button("Pagina pulsante"), "page1Button");
          page1Button.addMouseListener(this);

          add(page2Label = new Label("Pagina etichetta"), "page2Label");
          page2Label.addMouseListener(this);

          Panel panel = new Panel();
          panel.setLayout(new BorderLayout());
```

Figura 13.13
L'applet ThreePagesApplet.

```
            panel.add(page3Text = new TextArea("Pagina composta"));
            page3Text.addMouseListener(this);
            panel.add(page3Top = new Button("Pulsante superiore"), "North");
            page3Top.addMouseListener(this);

            panel.add(page3Bottom = new Button("Pulsante inferiore"), "South");
            page3Bottom.addMouseListener(this);

            add(panel, "panel");
        }

        public void mouseClicked(MouseEvent e) {
            layout.next(this);
        }

        public void mouseEntered(MouseEvent e) { }
        public void mouseExited(MouseEvent e) { }
        public void mousePressed(MouseEvent e) { }
        public void mouseReleased(MouseEvent e) { }

}
```

Quando l'utente fa clic su un componente nell'applet, viene mostrato il componente successivo nel layout. Ciò avviene richiamando il metodo `next()` sul layout. I metodi per il posizionamento per `CardLayout` sono:

- ✔ `first(Container)`
- ✔ `next(Container)`
- ✔ `previous(Container)`
- ✔ `last(Container)`

Questi metodi permettono di spostarsi rispettivamente al primo componente del layout, a quello successivo, a quello precedente e all'ultimo.

Tutti questi metodi richiedono che venga specificato un contenitore, in quanto è possibile utilizzare lo stesso gestore di layout contemporaneamente per più contenitori. Di norma, a meno che vi siano buoni motivi per non farlo, si crea un gestore di layout separato per ogni contenitore. In questo modo risulta più semplice rendere modulare il codice. Tuttavia, se si desidera minimizzare il numero totale di oggetti in un'applicazione di grandi dimensioni, è possibile fare in modo che diversi contenitori condividano un unico gestore di layout.

Quando nel contenitore vengono aggiunti componenti, si specifica una stringa che il gestore `CardLayout` utilizza per identificare il controllo. Invece di richiamare i metodi di posizionamento, ad esempio `next()` e `first()`, si può passare al componente con l'etichetta specificata richiamando il metodo `show(Container, String)` sul layout. Ad esempio, se si desidera passare alla seconda pagina nell'applet `ThreePagesApplet`, è possibile utilizzare la seguente chiamata:

```
layout.show(this, "page2Label");
```

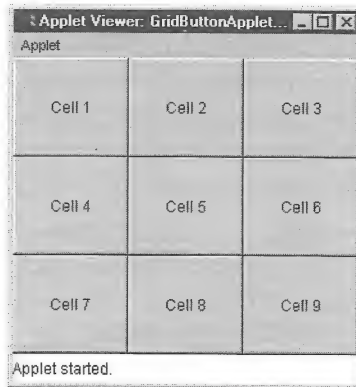
Il gestore GridLayout

Il gestore `GridLayout` viene utilizzato per disporre i componenti in una griglia di celle uniformemente distanziate. Il costruttore predefinito crea una griglia con una colonna per componente, ma è possibile utilizzare altri costruttori per specificare il numero esatto di righe e di colonne desiderato e anche la spaziatura tra le celle. Come si può vedere nella Figura 13.14, il gestore `GridLayout` riempie le celle da sinistra a destra e dall'alto verso il basso.

Il Listato 13.9 include il codice utilizzato per creare questo applet, che si trova anche nel CD-ROM allegato al libro.

Listato 13.9 *L'applet GridButtonApplet.*

```
import java.awt.*;

public class GridButtonApplet extends java.applet.Applet {

    public void init() {
        setLayout(new GridLayout(3, 3));
        add(new Button("Cell 1"));
        add(new Button("Cell 2"));
        add(new Button("Cell 3"));
        add(new Button("Cell 4"));
        add(new Button("Cell 5"));
        add(new Button("Cell 6"));
        add(new Button("Cell 7"));
        add(new Button("Cell 8"));
        add(new Button("Cell 9"));
    }
}
```

Figura 13.14
L'applet GridButtonApplet.

Il gestore GridBagLayout

`GridBagLayout` è il gestore di layout più complesso e più flessibile nell'AWT. Una volta abituatisi a utilizzarlo, lo si farà ogni volta per sviluppare interfacce simili a un modulo. Tuttavia, è difficile da apprendere e ha molte opzioni.

La base del gestore `GridBagLayout` è una griglia. A differenza del gestore `GridLayout`, i componenti nel gestore `GridBagLayout` non sono costretti in un'unica cella di dimensioni regolari: ogni cella ha un *peso* orizzontale e verticale, che viene utilizzato per determinare la proporzione dello spazio che occupa come valore in virgola mobile a precisione doppia compreso tra `0` (non occupa lo spazio disponibile) e `1` (occupa tutto lo spazio disponibile). Inoltre, ogni componente può occupare più celle. Vi sono numerosi altri fattori che controllano il posizionamento e le dimensioni dei componenti in un gestore `GridBagLayout`; questi possono essere specificati per mezzo di un oggetto `GridBagConstraints`, che viene aggiunto al layout contemporaneamente al componente e che specifica i fattori relativi. La classe `GridBagConstraints` contiene i seguenti dati membro.

- ✔ `gridx`, `gridy`: queste variabili specificano le coordinate delle celle per l'angolo a nord-ovest del componente. È possibile utilizzare il valore predefinito `GridBagConstraints.RELATIVE` per specificare un offset orizzontale o verticale di `1` dalla cella dell'ultimo componente aggiunto.

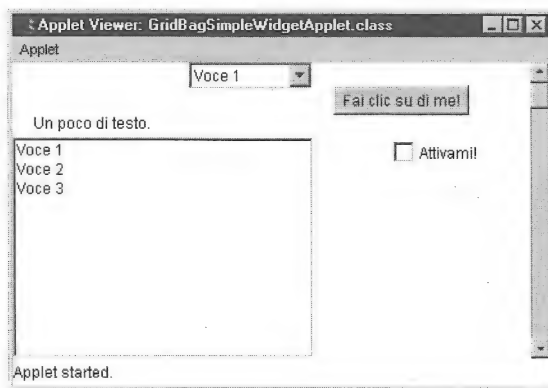
- ✔ `gridwidth`, `gridheight`: queste variabili specificano il numero di celle orizzontali e verticali che il componente occupa. È possibile utilizzare `GridBagConstraints.REMAINDER` per occupare tutte le celle rimanenti in una riga o in una colonna.

- ✔ `fill`: se il componente è più piccolo della cella o delle celle che occupa, questa variabile determina se, ed eventualmente come, modificare le dimensioni del componente in modo che occupi tutta l'area disponibile. Gli argomenti validi sono `GridBagConstraints.NONE` (non modificare le dimensioni), `GridBagConstraints.HORIZONTAL` (utilizzare tutta la larghezza), `GridBagConstraints.VERTICAL` (utilizzare tutta l'altezza) e `GridBagConstraints.BOTH` (utilizzare tutto lo spazio).

- `anchor`: se il componente è più piccolo della cella o delle celle che occupa, questa variabile determina su quale angolo o lato dell'area della cella inserire il componente. Gli argomenti validi sono `GridBagConstraints.CENTER` (l'argomento predefinito), `GridBagConstraints.NORTH`, `GridBagConstraints.EAST`, `GridBagConstraints.SOUTH`, `GridBagConstraints.WEST`, `GridBagConstraints.NORTHWEST`, `GridBagConstraints.NORTHEAST`, `GridBagConstraints.SOUTHEAST` e `GridBagConstraints.SOUTHWEST`.
- `weightx`, `weighty`: queste variabili vengono utilizzate per determinare il peso di una cella o quanto dello spazio in eccesso viene aggiunto alla riga o alla colonna. Se non si specifica il peso almeno di una riga e di una colonna, i componenti vengono disposti al centro, in quanto un peso uguale a `0` (il valore predefinito) aggiunge tutto lo spazio in eccesso ai lati.
- `Insets`: l'oggetto `Insets` specifica la distanza in pixel tra i bordi del componente e i bordi della cella o delle celle che occupa.
- `ipadx`, `ipady`: queste variabili controllano la spaziatura orizzontale e verticale tra i bordi del componente e quelli della cella o delle celle che occupa.

Nonostante le variabili `ipadx` e `ipady` siano ancora supportate in `GridBagConstraints` e dal gestore di layout, in realtà sono superate dalla variabile `Insets` e non è più consigliabile utilizzarle.

Per aggiungere un componente a un gestore `GridBagLayout`, è necessario innanzitutto impostare i dati membro appena elencati per un oggetto `GridBagConstraints` e quindi aggiungere il componente nel contenitore. L'applet `GridBagSimpleWidgetApplet` incluso nel Listato 13.10 è un esempio di utilizzo di un gestore `GridBagLayout` per controllare il layout dei componenti creati nell'applet `SimpleWidgetApplet`, presentato precedentemente in questo capitolo (Figura 13.15). L'applet `GridBagSimpleWidgetApplet` è incluso anche nel CD-ROM allegato al libro.

Figura 13.15 *L'applet GridBagSimpleWidgetApplet.*

Listato 13.10 *L'applet GridBagSimpleWidgetApplet.*

```
import java.awt.*;

public class GridBagSimpleWidgetApplet extends java.applet.Applet {

     public void init() {

          GridBagConstraints gbc;
          setLayout(new GridBagLayout());

          Button button = new Button("Fai clic su di me!");
          Checkbox checkbox = new Checkbox("Attivami!");
          Choice choice = new Choice();
          Label label = new Label("Un poco di testo.");
          List list = new List();
          Scrollbar scrollbar = new Scrollbar();

          choice.add("Voce 1");
          choice.add("Voce 2");
          choice.add("Voce 3");

          list.add("Voce 1");
          list.add("Voce 2");
          list.add("Voce 3");

          gbc = new GridBagConstraints();
          gbc.gridx = gbc.gridy = 0;
          gbc.anchor = GridBagConstraints.SOUTH;
          gbc.weightx = gbc.weighty = 1.0;
          add(label, gbc);

          gbc = new GridBagConstraints();
          gbc.gridx = 0;
          gbc.fill = GridBagConstraints.BOTH;
          gbc.gridwidth = 2;
          gbc.gridheight = 3;
          gbc.weightx = 1.0;
          gbc.weighty = 3.0;
          add(list, gbc);

          gbc = new GridBagConstraints();
          gbc.gridy = 0;
          gbc.fill = GridBagConstraints.HORIZONTAL;
          gbc.anchor = GridBagConstraints.NORTH;
          gbc.weightx = gbc.weighty = 1.0;
          add(choice, gbc);

          gbc = new GridBagConstraints();
          gbc.gridy = 0;
          gbc.weightx = gbc.weighty = 1.0;
          add(button, gbc);

          gbc = new GridBagConstraints();
          gbc.gridy = 2;
          gbc.anchor = GridBagConstraints.NORTHEAST;
          gbc.weightx = gbc.weighty = 1.0;
          add(checkbox, gbc);
```

```
        gbc = new GridBagConstraints();
        gbc.gridy = 0;
        gbc.gridheight = GridBagConstraints.REMAINDER;
        gbc.fill = GridBagConstraints.VERTICAL;
        gbc.anchor = GridBagConstraints.EAST;
        gbc.weightx = gbc.weighty = 1.0;
        add(scrollbar, gbc);

    }
}
```

Layout personalizzati

Se non si ottiene il layout desiderato per un contenitore, è possibile scrivere un proprio gestore di layout. È necessario implementare l'interfaccia `LayoutManager` o l'interfaccia `LayoutManager2` e fornire i relativi metodi. `LayoutManager2` è un'interfaccia con una minima estensione di `LayoutManager`, con lo scopo di assegnare componenti al layout sulla base di un oggetto `Constraints`, ad esempio un oggetto `GridBagConstraints` per il gestore `GridBagLayout` o l'oggetto stringa `"North"` per il gestore `BorderLayout`.

Classi di utilità

Le classi di utilità rappresentano strutture e risorse utili, molte delle quali sono fondamentali per la programmazione della grafica e possono agire come proprietà dei controlli. Le classi di utilità sono raggruppate nel seguente modo.

- ✔ **Vettori:** `Dimension`, `Insets`, `Point`, `Polygon`, `Rectangle`, `Shape`.
- ✔ **Colori:** `Color`, `SystemColor`.
- ✔ **Risorse:** `Cursor`, `Font`, `FontMetrics`, `Graphics`, `Image`, `PrintGraphics`, `PrintJob`, `Toolkit`.
- ✔ **Utilità per la manipolazione di risorse:** `MediaTracker`.

Poiché le classi dei vettori sono intese principalmente per memorizzare e richiamare informazioni sulla posizione, tutti i loro campi sono pubblici. A un componente viene aggiunto un oggetto `Insets` per specificare la spaziatura tra il componente e quello intorno; gli oggetti `Insets` sono gestiti dal gestore di layout.

Le classi dei colori incapsulano in modo efficace fino a 16.777.216 colori, che corrispondono ai colori supportati dalla maggior parte dei sistemi operativi attuali. La classe `SystemColor` può essere utilizzata per richiamare informazioni sui colori correnti del desktop, inclusi quelli del testo e quelli in primo piano e sullo sfondo nelle finestre.

Gli oggetti `Cursor` incapsulano la forma del puntatore del mouse. Possono essere di diversi tipi predefiniti, incluso il cursore predefinito (di norma una freccia), un cursore di inserimento del testo, un cursore di attesa per indicare che è in corso un compito che non può essere interrotto, un cursore a forma di mano per selezionare elementi correlati (ad esempio

i collegamenti ipertestuali), un cursore a croce per posizionare le immagini e diversi cursori di ridimensionamento. Per il cursore non è possibile definire forme personalizzate, in quanto le forme sono determinate in base ai cursori della piattaforma.

I tipi di carattere e le dimensioni permettono di descrivere i caratteri in Java. Ogni implementazione di Java ha almeno sei caratteri: Courier, Dialog, DialogInput, Helvetica, Times Roman e ZapfDingbats. Il carattere utilizzato dalla piattaforma varia in base alla piattaforma specifica. Ad esempio, per il carattere Dialog in Windows viene utilizzato MS Sans Serif.

La classe `Toolkit` incapsula una serie di strumenti per le finestre nella piattaforma sottostante che viene utilizzata per creare oggetti peer (si veda il paragrafo: "Oggetti peer" più avanti in questo capitolo), per restituire un elenco dei caratteri e dei lavori di stampa correnti e per permettere di richiedere informazioni quali la risoluzione e le dimensioni dello schermo.

Caricamento di immagini

La classe `Image` rappresenta un'immagine. Vi sono due modi standard per caricare un'immagine in Java, a seconda se si sviluppa un'applet o un'applicazione. Per gli applet si utilizza il metodo `getImage(`*`URL`*`)` sull'applet stesso. Per le applicazioni, si utilizza `Toolkit.getDefaultToolkit()` o `Toolkit.getImage(`*`URL`*`)`. Attualmente sono supportate dall'AWT immagini nei formati GIF e JPEG. Quando si richiama `getImage()`, viene restituito immediatamente un riferimento valido a un'immagine, ma l'immagine viene caricata in un thread separato. La classe `MediaTracker` può essere utilizzata per tenere traccia del processo di caricamento delle immagini; ad esempio, se prima di visualizzare un frame si desidera aspettare finché un'immagine è stata completamente caricata, è possibile utilizzare la classe `MediaTracker` nel seguente modo:

```
Frame f = new Frame("Test MediaTracker");
MediaTracker mt = new MediaTracker(f);
Image img = Toolkit.getDefaultToolkit().getImage(new
                                      URL("http://mioserver.com/miaimmagine.gif"));
mt.addImage(img, 0);
mt.waitForAll();
f.show();
```

Stampa

La classe `PrintJob` e l'interfaccia `PrintGraphics` sono utilizzate per fornire l'API di stampa. Stampare in Java è molto semplice: innanzitutto si inizia un lavoro di stampa richiamando `getPrintJob(`*`Frame, Stringa, Proprietà`*`)` su un oggetto `Toolkit` valido. L'oggetto `Properties` permette di assegnare valori predefiniti per il lavoro di stampa specifici per la piattaforma (e a volte per la stampante). Il metodo `getPrintJob()` visualizza all'utente una finestra di dialogo e pertanto l'oggetto `PrintJob` restituito sarà completo di tutte le proprietà specificate dall'utente. Successivamente è possibile richiamare `getGraphics()` sul `PrintJob`; questo metodo fornisce un oggetto `Graphics` che implementa l'interfaccia `PrintGraphics` e che può essere passato al metodo `paint()` o `print()`, oppure utilizzato da metodi grafici.

Quando si è terminato con il contesto grafico, lo si può inviare alla stampante richiamando dispose(). Ogni contesto grafico rappresenta una pagina, pertanto è possibile stampare diverse pagine effettuando diverse chiamate a getGraphics(). È possibile determinare gli attributi e la risoluzione delle pagine con i metodi getPageDimensions() e getPageResolution() sul PrintJob. Si noti che l'avvio di un processo di stampa è un'operazione limitata da problemi di sicurezza: gli applet non fidati non possono stampare materiale potenzialmente pericoloso sulla stampante di loro scelta.

Oggetti peer

I controlli semplici (e le classi Panel e Canvas) della libreria di classi dell'AWT utilizzano un meccanismo detto "peer" per determinare gli aspetti visivi del componente. Un *peer* è un oggetto creato dalla piattaforma nativa a finestre gestito dalla macchina virtuale di Java e ha lo stesso aspetto degli altri componenti non Java della stessa piattaforma. Ad esempio, un elenco Choice istanziato su un desktop Motif crea un componente in rilievo con una piccola lineetta tridimensionale che apre un menu tridimensionale centrato sull'elemento correntemente selezionato, ma nel desktop di Windows 95, lo stesso elenco Choice crea una casella rientrata (come un campo di testo) con un pulsante che mostra una freccia rivolta verso il basso che apre un menu piatto sotto il pulsante.

Il vantaggio dell'utilizzo dei meccanismi peer è che gli utenti di una particolare piattaforma a finestre possono sempre ottenere componenti della GUI che ricordano quelli a cui sono abituati; inoltre i peer permettono il semplice trasferimento di dati da e per gli Appunti del sistema, in quanto gli oggetti creati sono gestiti dalla piattaforma e non dalla macchina virtuale di Java.

Tuttavia, gli oggetti peer presentano diversi svantaggi. Innanzitutto l'area coperta dai componenti che si basano sui peer è sempre riempita con il colore di sfondo del componente. Se si desidera un'immagine di sfondo per un'applicazione, l'immagine sarà sempre coperta dai componenti basati sui peer aggiunti. Secondariamente, la piattaforma a finestre necessita di ulteriore supporto sotto forma di risorse di sistema per gestire i peer creati e le chiamate dalla macchina virtuale di Java al sistema operativo per sincronizzare le operazioni sui componenti con i peer correlati possono aumentare il tempo di elaborazione necessario.

Vi è comunque un'alternativa all'utilizzo degli oggetti peer. Anziché creare sottoclassi per i componenti da Canvas e da Panel, è possibile derivarle direttamente da Component e da Container, scrivendo metodi paint() nello stesso modo, ma senza il supporto necessario per mantenere un peer. Ad esempio, è possibile derivare la classe ImageButton, descritta nel paragrafo precedente relativo alla classe Canvas, da Component invece che da Canvas. Con questo approccio, qualunque cosa visibile nel contenitore di ImageButton traspare attraverso il pulsante, a meno che prima non si scelga di riempire il riquadro del contenitore con il colore di sfondo del pulsante.

Un altro vantaggio che si ottiene scrivendo i propri componenti non peer è che si ha un buon controllo sul loro aspetto. È anche possibile correggere errori nelle implementazioni

specifiche per le piattaforme delle classi peer, su cui altrimenti non si avrebbe controllo in Java. Ad esempio, un componente `List` che si basa sui peer in una piattaforma Windows 95 ha sempre un bordo nero. Se l'utente desidera uno sfondo nero, non sarà in grado di vedere il bordo del controllo. È possibile risolvere questo problema nei componenti non peer esaminando i colori del sistema, descritti nel paragrafo: "Classi di utilità" precedentemente in questo capitolo, e disegnando il bordo con un colore appropriato.

La Sun intende distribuire una serie più ampia di componenti semplici non peer (chiamati anche "pesi leggeri") nella prossima versione dell'AWT, che dovrebbe inoltre avere, come standard, funzionalità più complesse di rappresentazione e di trascinamento con il mouse.

Riepilogo

L'AWT di Java contiene una collezione di componenti e diversi modi per disporli, che possono essere utilizzati così come sono o ampliati per fornire un'interfaccia utente grafica per tutte le piattaforme, completa e uniforme. La presentazione di informazioni agli utenti e la capacità di rispondere ai numerosi modi in cui gli utenti manipolano l'applicazione è un compito complesso e impegnativo, che può essere reso notevolmente più semplice per mezzo dell'incapsulamento delle funzionalità e delle proprietà di visualizzazione negli oggetti dell'AWT. Utilizzando l'AWT è possibile creare semplici applet o applicazioni complete di tutte le funzionalità che possono essere eseguiti su piattaforme software e hardware diverse.